一、引言

年年涨价，仍然是供不应求。茅台、五粮液这样的高端白酒在国内一次次的拨动着市场的神经。自 2011 年元旦开始，茅台全国涨价 20%，贵州茅台董事长袁仁国强调了严苛的限价令，终端价格（市场零售价）不得超过 959 元，但是市场上的茅台酒仍然一瓶难求，市场终端价格也早已超出限价的一倍，达到 1599 元左右，而仍然是难以原价购买。

追溯到上世纪 80 年代，茅台酒曾经 8 元一瓶，20 余年过去，价格上涨了 187 倍，与此同时，陈年茅台连续拍上数百万的高价。对于茅台价格的各种质疑一直层出不穷。那么茅台相对价格到底是涨还是跌？在股票市场上茅台的表现又是如何？到底是什么因素导致茅台股票越涨越畅销？我们将通过对贵州茅台公司的股票进行预测和对股吧中网友的评论进行情感分析，这两个途径来寻找答案。

二、LSTM 股票预测

注：股票预测部分一共训练了 Model_A 与 Model_B 两个模型。训练两个模型的原因：一是想观察特征数对模型训练结果的影响，二是两个模型在实际应用方面的不同。模型 A 是以 12 个特征来预测'收盘价'这一特征，在实际应用方面只能预测出未来一天的股票价格，而模型 B 是只用'收盘价'来预测'收盘价'，能预测出未来自定义天数的股票价格走势。下面介绍的是模型 B 训练与应用的主要步骤。

思路：通过近几年贵州茅台的历史股票交易数据训练出 LSTM 模型，运用训练的模型预测未来自定义天数的股票价格走势，作为股票买入与卖出的参考之一。

主要步骤：

第一步，数据爬取

从网易财经网站爬取下来贵州茅台股票两年的历史交易数据，爬取的过程用到了 Python selenium 包中的 Webdriver，Webdriver 是一个用来进行复杂重复的 web 自动化测试工具，通常用于编写 web 应用的自动化测试，也可以运用到我们的爬虫工作中。本次爬取用到 Chrome 浏览器配合 Webdriver 中的 clear()、send_keys()和 click()（清除、赋值、模拟点击）方法。为防止某些时候网页加载过慢，Webdriver 没有在 DOM 中找到元素，我们设置了一个隐式等待（implicitly_wait()），此外，我们还用到了 ChromeOptions()来更改 Chrome 的默认下载地址，使要下载的文件能正好下载到我们的项目文件夹中。为防止出现文件还未下载完成浏览器就关闭的情况，我们在方法的最后加上了 time.sleep()方法。下图为数据爬取部分用到的主要方法：

```
def download_historical_transaction_data(company_code,start_time,end_time,download_path):
    options = webdriver.ChromeOptions()
    prefs = {'download.default_directory': download_path}
    options.add_experimental_option('prefs', prefs)
    driver = webdriver.Chrome(options=options)

    driver.implicitly_wait(30)  # 隐式等待
    data_url = "http://quotes.money.163.com/trade/lsjysj_" + str(company_code) + ".html#01b07"
    driver.get(data_url)
    driver.find_element_by_id("downloadData").click() #找到"下载数据"并点击
    driver.find_element_by_name("date_start_value").clear() #清除"起始日期"输入框
    driver.find_element_by_name("date_start_value").send_keys(start_time) #"起始日期"输入框重新赋值
    driver.find_element_by_name("date_end_value").clear() #清除"截止日期"输入框
    driver.find_element_by_name("date_end_value").send_keys(end_time) #"截止日期"输入框重新赋值
    driver.find_element_by_xpath('//div[@class = "align_c"]/a').click() #找到"下载"并点击
    time.sleep(5)
    driver.quit()
```

第二步，数据预处理

将爬取的数据读入，去掉‘收盘价’外的其他特征列，简单处理之后进行归一化。归一化用到了 sklearn 中的 MinMaxScaler。数学原理如下：

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min

（min，max = feature_range）

feature_range：默认为(0,1)，期望的传唤数据范围。

我们的目标是用 19 天的股票价格来预测第 20 天的股票价格，于是，对数据进行归一化后，又对数据做了一个类似窗口大小为 20，步长为 1 的滑动窗口处理。将 n 条数据分成 n-19 组，每组元素个数为 20，每组前 19 个元素的特征值作为 x，第 20 个元素的‘收盘价’特征作为 y，用以模型训练。数据集的切分我们选择了 80%的数据作为训练集，剩下的为测试集，之后使用 torch.Tensor()方法将数据集转化为张量（一个可以运行在 GPU 上的多维数据，加快计算效率）。下图为数据预处理部分用到的主要方法：

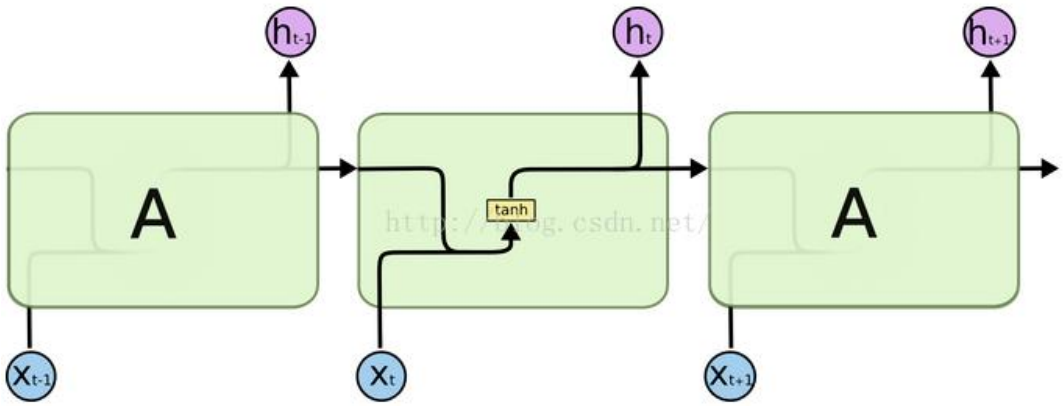
```
def data_normalization(data):
    price = data.copy()
    scaler = MinMaxScaler(feature_range=(-1, 1))
    price.iloc[:, 0] = scaler.fit_transform(price.iloc[:, 0].values.reshape(-1, 1))

    return price, scaler
```

```python
def split_data(stock, lookback):
    data_raw = stock.to_numpy()
    data_set = []

    for index in range(len(data_raw) - lookback):
        data_set.append(data_raw[index:index + lookback])

    data = np.array(data_set)
    test_set_size = int(np.round(0.2 * data.shape[0]))
    train_set_size = data.shape[0] - (test_set_size)

    x_train = data[:train_set_size, :-1, :]
    y_train = data[:train_set_size, -1, 0:1]

    x_test = data[train_set_size:, :-1, :]
    y_test = data[train_set_size:, -1, 0:1]

    return [x_train, y_train, x_test, y_test]

    # 将数据集转化为张量（一个可以运行在GPU上的多维数据，加快计算效率）

def data_transform(x_train, y_train, x_test, y_test):
    x_train = torch.Tensor(x_train)
    x_test = torch.Tensor(x_test)
    y_train_lstm = torch.Tensor(y_train)
    y_test_lstm = torch.Tensor(y_test)

    return [x_train, x_test, y_train_lstm, y_test_lstm]
```

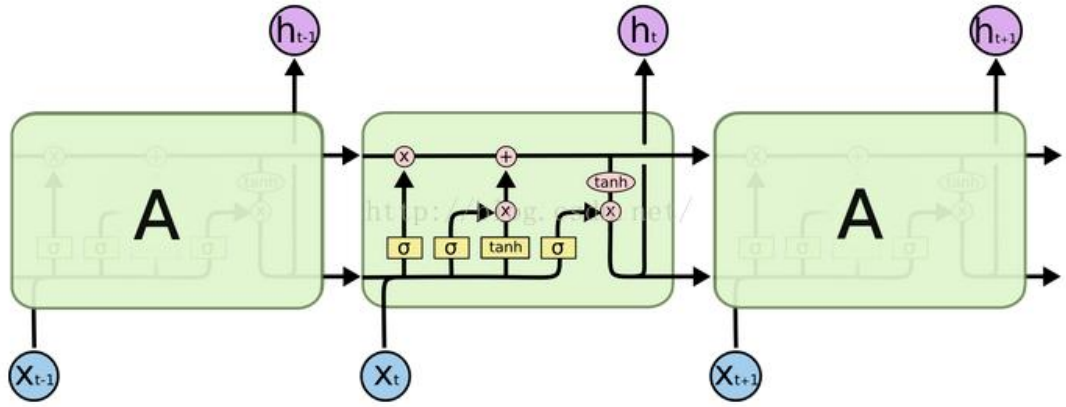
第三步，模型构建训练

股票的价格波动属于时间序列，我们选择使用 LSTM-长短期记忆递归神经网络来进行模型的构建。LSTM 可以被简单理解为是一种神经元更加复杂的 RNN 循环神经网络(Recurrent Neural Network)，能够解决一般循环神经网络中普遍存在的长期依赖问题，使用 LSTM 可以有效的传递和表达长时间序列中的信息并且不会导致长时间前的有用信息被忽略（遗忘）。与此同时，LSTM 还可以解决 RNN 中的梯度消失/爆炸问题。下图可以看出 RNN 与 LSTM 的区别：

RNN



LSTM



其中：

我们的 LSTM 模型是基于 PyTorch 框架构建的，张量的维度（input_dim）为 1，隐藏层神经元个数(hidden_dim)设置为 32, LSTM 图层数(num_layers)设置为 2, 输出维度(output_dim)为 1。模型训练用得到的损失函数为 MSE（均方差损失）

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(yi - f(x_i))^2$$

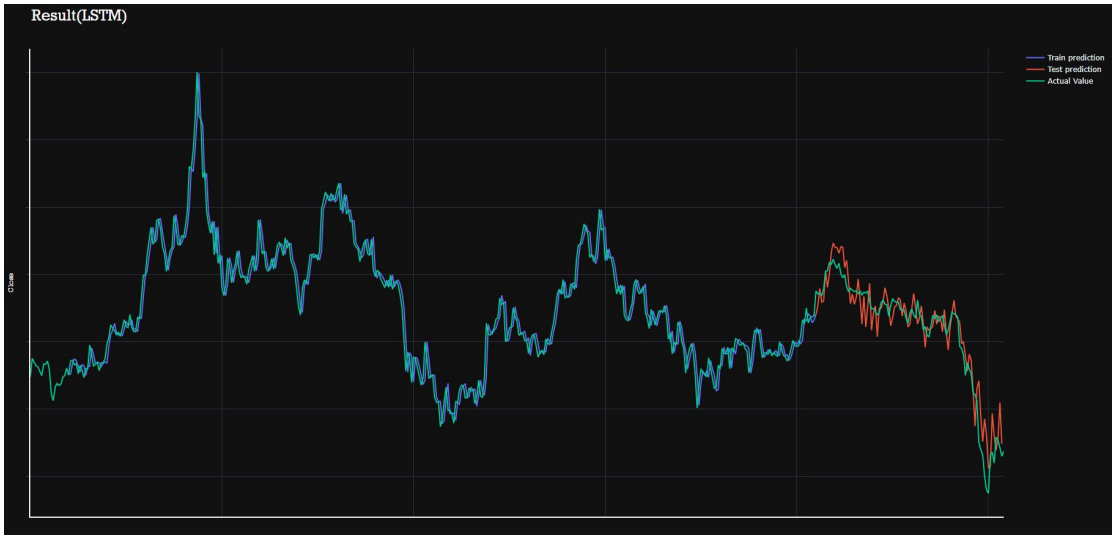
, 优化器为 Adam。经过 1000 期(num_epochs = 1000）的训练后，loss 的值能够降到 0.00010 上下。下面为主要代码及训练结果：

```python
class LSTM_B(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM_B, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out
```

```python
def model_training(input_dim,hidden_dim,num_layers,output_dim,num_epochs,x_train,y_train_lstm):
    model = LSTM_B(input_dim=input_dim, hidden_dim=hidden_dim, num_layers=num_layers, output_dim=output_dim)
    criterion = torch.nn.MSELoss()  # 定义均方差损失函数
    optimiser = torch.optim.Adam(model.parameters(), lr=0.01)  # 定义优化器

    hist = np.zeros(num_epochs)
    start_time = time.time()

    for t in range(num_epochs):
        y_train_pred = model(x_train)

        loss = criterion(y_train_pred, y_train_lstm)
        print('Epoch', t, 'MSE:', loss.item())
        hist[t] = loss.item()

        optimiser.zero_grad()
        loss.backward()     # loss反向传播
        optimiser.step()    # 梯度更新

    training_time = time.time() - start_time
    print('Training time :{}'.format(training_time))

    return model
```

下图为训练结果的可视化:

```
Epoch 992 MSE: 0.00015215187158901244
Epoch 993 MSE: 0.00013201794354245067
Epoch 994 MSE: 0.00010498513438506052
Epoch 995 MSE: 0.00012724936823360622
Epoch 996 MSE: 0.00010625895811244845
Epoch 997 MSE: 9.760275133885443e-05
Epoch 998 MSE: 9.821782441576943e-05
Epoch 999 MSE: 9.581514314049855e-05
Training time :50.108288288116455
```



最后将训练好的模型保存。

第四步，模型验证

模型验证用到了 RMSE（均方根误差），来衡量观测值同真值之间的偏差，

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(Y_i - f(x_i))^2}$$

在训练集和测试集上得出的结果分别为为 Train Score: 5.62 RMSE，Test Score: 55.26 RMSE。下图为主要代码：

```
def model_validation(y_train,y_train_pred,y_test,y_test_pred):
    trainScore = math.sqrt(mean_squared_error(y_train[:, 0], y_train_pred[:, 0]))
    print('Train Score: %.2f RMSE' % (trainScore))

    testScore = math.sqrt(mean_squared_error(y_test[:, 0], y_test_pred[:, 0]))
    print('Test Score: %.2f RMSE' % (testScore))
```

第五步，模型应用

读入最近 19 天的股票交易数据后，同样进行简单处理与归一化，然后读取保存好的模型，直接对 x 进行预测，得出 y。关键步骤为：将得出 y 加入到数据集中组成新的 x，以进行自定义天数数据的预测。下图为主要代码：

```python
def stock_pre(stock,lookback,pre_days,model):
    data_raw = stock.to_numpy()
    data_set = []
    data_pre = []

    data_set.append(data_raw[0:0 + lookback])
    data_set_array = np.array(data_set)

    for i in range(pre_days):
        i += 1
        x = data_set_array[-1:, :, :]
        x = data_transform(x)

        y = model(x)
        data_pre.append(y)

        y = y.detach().numpy()

        data_raw = data_raw.tolist()
        data_raw.append(y[0])        #将预测的y加入data_raw以组成新的x
        data_raw = np.array(data_raw)
        data_set.append(data_raw[i:i + lookback])
        data_set_array = np.array(data_set)

    return data_pre
```

运用模型我们最后得出了从 2022-11-11 开始往后一个月贵州茅台股票价格的变化趋势。下图为结果的可视化：

三、情感分析

情感分析的主要目的是通过获取炒股用户在相应公司股票的栏目下的评论，对评论所含的情感进行分类，从而获得用户推荐购入此类股票和用户不推荐购买此类股票的信号，对于

获取到的分析结果，可以通过分词来判断在用户买入股票的评论中，蕴含那些高频词可以代表相应的情绪，对不同的词汇进行频率编纂和分析，用可视化的手段直观反映用户的整体情绪，从而帮助使用者对于是否买入股票有一定的了解。
团队事先获取了 900 条贵州茅台股票的评论，对其进行人工分类，作为训练模型的数据集，之后利用朴素贝叶斯算法建模拟合股票评论和情感标签从而得到模型

朴素贝叶斯实践思路：

1. 将导入语句进行分词
2. 去除导入语句停用词
3. 将分词作为特征、将情感标签作为类别标记，把两者使用训练方法拟合

朴素贝叶斯原理：

贝叶斯模型需要传入两种参数，其一为数据特征，其二为类别标记。在该场景下可用 y={Pos_tone,Neg_tone}作为类别标记。X={segword_1,segword_2,segword_3…,segword_n}作为传入特征。往往词语之间具有较高的独立性，因此我们可以采用属性条件独立性假设。根据上述条件，以计算机及语调为实例演示计算过程：

$$P(pos_{tone}|x) = \frac{\widehat{P}(Pos\_tone)\widehat{P}(x|Pos\_tone)}{P(x)}$$ ----------注

*先验概率 P(Pos_tone)与条件概率 P(x|Pos_tone)均使用拉普拉斯修正，以防止新实例中携带训练集中不存在的信息时，将概率值抹除。即若令 N 表示训练集 D 的类别数，Ni 为第 i 个特征可取值的个数。

$$\widehat{P}(Pos\_tone) = \frac{D_{pos\_tone}+1}{D+N}\ ,\ \widehat{P}(x|Pos\_tone) = \frac{D_{pos\_tone}\,xi+1}{D_{pos\_tone}+Ni}$$

由于本次数据中不包含连续型特征，故不讲解条件概率密度函数的算法。

随后计算：

$$P_{xi|Pos\_tone} = P(x_i = segword_i|y = Pos\_tone)$$ ①

$$P_{xi|Nos\_tone} = P(x_i = segword_i|y = Neg\_tone)$$ ②

① 式得到所有分词是积极语调的概率值

② 式得到所有分词是消极语调的概率值

判断实例的方法：设实例粉刺处理后特征包含为$\hat{x} = \{word1, word2, \ldots., wordn\}$

$$P(y = pos\_tone) \times \prod_{i=1}^{n} Pxi|pos\_tone = P_{postive}$$

$$P(y = neg\_tone) \times \prod_{i=1}^{n} Pxi|neg\_tone = P_{negative}$$

若$P_{postive} > P_{negative}$ 则为积极语调，反之则为消极语调。

四、词云图分析

使用爬虫程序和情感分析获得了具有情感分析标签的评论数据后对数据进行处理：

1、 数据清洗及预处理：利用 pandas 对数据进行导入后使用 request 删除字母和限定的名词
2、 分词：分词部分使用 jieba 的 posseg 接口，得到含有词与词性的二维列表，同时构造句子 id 和句子所含词的个数的变量
3、 合并：将构造的词列、词性列、id 列、情感标签列合并为一个 DataFrame 便于后续处理
4、 分割语料库：将感情标签作为分割依据分为两种数据框
5、 分组汇总：统计重复词汇并计数，并保存到新的变量

作图：引用 matplotlib 和 wordcloud，构建实例设置相应的参数，拟合数据后画图
总词汇词云图：

积极词云图：

消极词云图：

五、测试及自我评价

LSTM 模型的拟合效果总体不错，但是有些许的滞后性，这也是 LSTM 在股票预测方面常见的问题，主要原因是影响因素过多，特征量不足，希望日后能够找到有效的解决办法。另外，训练出的模型在测试集上的 RMSE 值偏高，模型还有进一步优化的空间。不过，从模型应用的结果来看，模型具有一定的应用价值。

情感分析的编程部分在项目初期使用的神经网络和逻辑回归的模型分析的、但是随着机器学习和其他必修课程的推进团队在 sklearn 中了解到了朴素贝叶斯这种算法，可以说是一见钟情，简单的框架、高效的运行效率、明了的结果，团队在建模过程屡屡受阻，最终选择了使用 snownlp 这种工具帮忙搭建贝叶斯模型，在考虑模型的多种条件下，过量的数据在测试时出现过拟合的情况泛化能力相对于少量数据反而减少了，因此团队选择了相对适中的数据量对模型进行训练。

词云图部分的分词过程让团队的 python 功底得到了检验，可以说是千辛万苦，在处理句子 id 的部分反复报错，只得依次找错一系列难题，只求句子的简洁度、易读性和运行的效率有所提高，最后的数据框合并完成才得以松一口气，词云的构图部分引用了 wordcloud 对于不同的词语和相应的词频进行了拟合得到了全部词语、消极词语、积极词语三张不同的词云图，方便后续的分析和总结，可以更直观的了解什么词语反映了什么样的情绪。

六、总结

从数据结果、人们评价、股票走势来看茅台股价一直在稳步提高。随着人们生活水平的提高，对于高档白酒的消费将不断增加。因此，凭借其“国酒”的身份地位，茅台酒一直处于供不应求的状态。茅台酒由于饥饿营销和资源稀缺性，不可能无限制的扩大产能。但是，茅台酒有一个生产周期，未来茅台产能的增长与扩大还是非常肯定的。从股票市场来看，股民对于茅台股票的抛售是看跌期权，短时间内的波动是必然的，但是总体来看未来一个月茅台的股票很大概率呈上涨趋势。